



L'allocazione dinamica della memoria

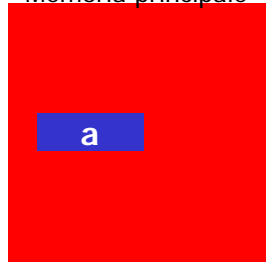


Fulvio CORNO - Matteo SONZA REORDA
Dip. Automatica e Informatica
Politecnico di Torino

I puntatori

Le variabili di tipo puntatore permettono di accedere alla memoria in modo indiretto.

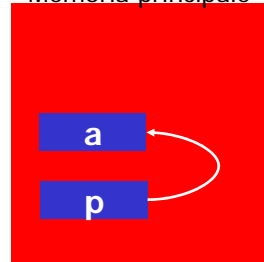
Memoria principale



```
int a;  
...  
a=10;
```

A.A. 2001/2002

Memoria principale



```
int a; int *p;  
...  
p=&a; *p=10;
```

APA - Memoria dinamica

2



Operatori * e &

L'operatore & permette di risalire dal nome di una variabile al suo puntatore:

```
p = &a;
```

L'operatore * permette di accedere alla cella di memoria referenziata da una variabile puntatore:

```
*p = 10;
```

A.A. 2001/2002

APA - Memoria dinamica

3



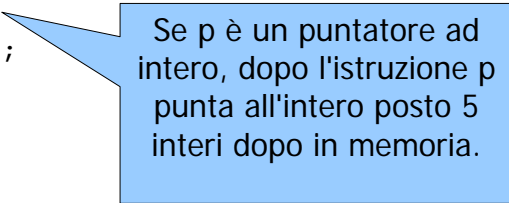
Operazioni sui puntatori

Assegnazione:

```
p = q; /* con un altro puntatore */  
p = NULL; /* con la costante NULL */
```

Incremento/decremento

```
p = p+5;  
p = p-10;  
p++;
```



Se p è un puntatore ad intero, dopo l'istruzione p punta all'intero posto 5 interi dopo in memoria.

A.A. 2001/2002

APA - Memoria dinamica

4



Usò dei puntatori

Scansione e azzeramento di un vettore

```
...  
int vett[N];  
int *p;  
  
...  
p=&vett[0];  
for (i=0; i<N; i++)  
    *p++=0;  
...
```



Puntatori a struct

Qualora una variabile p sia di tipo puntatore a struct, ad essa è applicabile l'operatore $->$:

$$p->\text{nome_campo} \equiv (*p).\text{nome_campo}$$



Esempio

```
struct scheda{
    int codice;
    char nome[20];
    char cognome[20];
};
struct scheda *p;
struct scheda vett[N];
...
p=vett;
for (i=0; i<N, i**)
{ p->codice=0;
  p++;
}
```

Equivale a
(*p).codice=0;



Puntatori e vettori

In C il nome di una variabile di tipo vettore coincide con il puntatore al primo elemento del vettore.

Quindi puntatori e nomi di vettori sono intercambiabili.



Esempio

Definizioni:

```
int vett[MAX];  
int *p;
```

Inizializzazione:

```
p = vett;           ≡      p=&vett[0];
```

Forme equivalenti:

```
vett[0]=10;        ≡      *p=10;  
vett[10]=25;       ≡      *(p+10)=25;  
vett[i]=0;         ≡      *(p+i)=0;  
*vett=27;          ≡      p[0]=27;
```



Allocazione statica

La memoria necessaria per le variabili globali in C resta occupata dal momento dell'attivazione del programma sino al suo termine.

Inoltre non è possibile dimensionare tale memoria secondo le esigenze della singola attivazione.

Si parla quindi di allocazione **statica** della memoria.



Esempio

```
#define MAX 1000  
  
struct scheda vett[MAX];
```

vett è dimensionato per eccesso, in modo da poter soddisfare in ogni caso le esigenze del programma, anche se questo usa un numero minore di elementi. Quindi in ogni caso vett occupa 42.000 byte.

A.A. 2001/2002

APA - Memoria dinamica

11



Esempio

Memoria principale



A.A. 2001/2002

APA - Memoria dinamica

12



Allocazione dinamica

Molti linguaggi di alto livello supportano l'allocazione **dinamica** della memoria.

Questo significa che il programmatore può inserire nel proprio codice delle chiamate a procedure di sistema che

- richiedono l'**allocazione** di un'area di memoria
- richiedono il **rilascio** di un'area di memoria.



Esempio 1

Il programma è in grado di determinare, ogni volta che è lanciato, di quanta memoria ha bisogno.

Viene allora chiamata una procedura di sistema, che provvede all'allocazione della memoria necessaria.



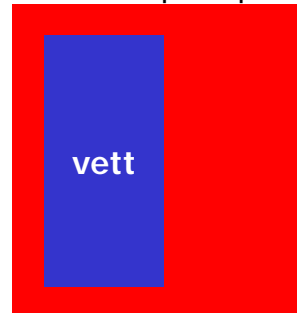
Esempio 1

Memoria principale



Caso 1: il programma
ha bisogno di **poca** memoria

Memoria principale



Caso 2: il programma
ha bisogno di **molta** memoria

A.A. 2001/2002

APA - Memoria dinamica

15



Esempio 2

Durante l'esecuzione, il programma ha bisogno di una quantità **variabile** di memoria.

Il programma usa ad ogni istante solo la memoria di cui ha bisogno, provvedendo periodicamente ad allocare o deallocare memoria.

In tal modo si permette ad eventuali altri processi che lavorano in parallelo sullo stesso sistema di meglio utilizzare la memoria disponibile.

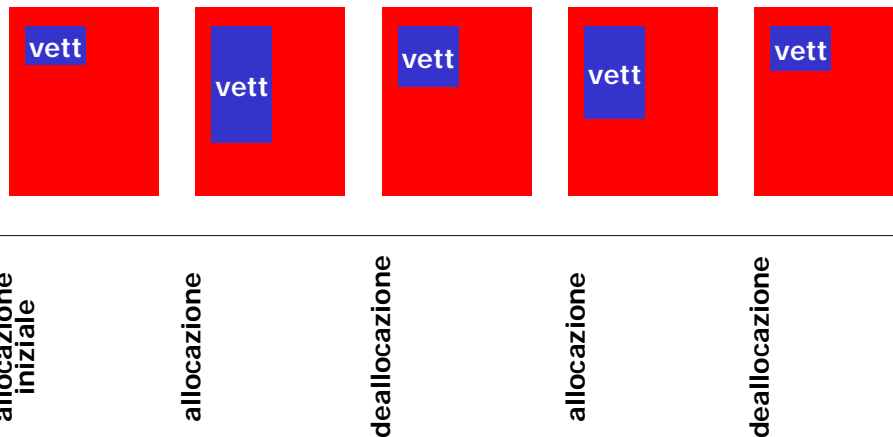
A.A. 2001/2002

APA - Memoria dinamica

16



Esempio 2



A.A. 2001/2002

APA - Memoria dinamica

17



malloc

Il C fornisce una procedura di libreria che permette di allocare dinamicamente la memoria necessaria.

```
void *malloc (int n);
```

Richiede al sistema operativo l'allocazione di una zona di memoria di dimensione (in byte) pari ad n , e ritorna il puntatore all'inizio della zona.

A.A. 2001/2002

APA - Memoria dinamica

18

Esempio

```
int *punt;  
int n;  
...  
punt = (int *)malloc(n);  
if (punt == NULL)  
{ printf ("Errore di allocazione\n");  
  exit();  
}  
...
```

Richiede l'allocazione di una zona di memoria di n byte.

Verifica che l'allocazione sia avvenuta regolarmente.

Trasforma il puntatore generico ritornato da malloc in un puntatore a int.

A.A. 2001/2002

APA - Memoria dinamica

19

Puntatori e malloc

Si può quindi:

- Allocare un vettore in modo dinamico, tramite la procedura `malloc`
- Usare il puntatore ritornato per accedere al vettore, come si farebbe con il nome di un vettore.

A.A. 2001/2002

APA - Memoria dinamica

20



Esempio

Si vuole scrivere una procedura `alloca` che

- Legge da tastiera un numero `n`
- Alloca un vettore di `n` elementi di tipo `struct scheda`
- Inizializza ogni elemento del vettore.

A.A. 2001/2002

APA - Memoria dinamica

21



Procedura `alloca`

```
int n;      /* variabile globale */
...
struct scheda *alloca (void)
{ int i; struct scheda *p;
  scanf("%d", &n);
  p=(struct scheda *)malloc(n*sizeof(struct scheda));
  if (p==NULL)
    return (NULL);
  for (i=0; i<n; i++)
  { p[i].codice=0;
    strcpy(p[i].nome, "");
    strcpy(p[i].cognome, "");
  }
  return (p);
}
```

A.A. 2001/2002

APA - Memoria dinamica

22



Allocazione dinamica di stringhe

In C le stringhe sono memorizzate sotto forma di vettori di caratteri, usando "\0" come carattere di fine stringa.

Quando si deve memorizzare una stringa di n caratteri si può quindi

- Usare un vettore allocato staticamente di lunghezza $N > n$ oppure
- Allocare dinamicamente un vettore di lunghezza $n+1$ byte.



Esempio

Si vuole scrivere una procedura `read` che legge da tastiera i dati relativi ad n schede, e li memorizza nel vettore precedentemente allocato.



Procedura read

```
int read (struct scheda *p)
{ int i, val; char nome[MAX], cogn[MAX];
  for (i=0, i<n; i++)
  { scanf ("%d %s %s\n", &val, nome, cogn);
    p[i].codice=val;
    p[i].nome=strdup(nome);
    if (p[i].nome == NULL)
      return (-1);
    p[i].cogn=strdup(cogn);
    if (p[i].cogn == NULL)
      return (-1);
  }
  return (0);
}
```

A.A. 2001/2002

APA - Memoria dinamica

25



Procedura strdup

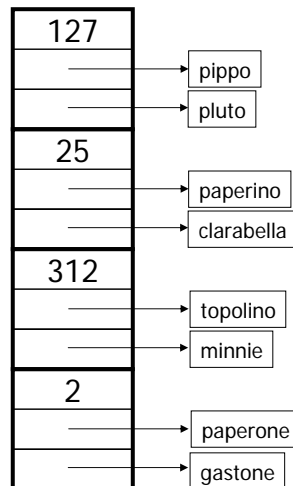
```
char *strdup (char *str)
{ int len; char *p;
  len=strlen (str);
  p=(char *)malloc(len);
  if (p==NULL)
    return (NULL);
  strcpy (p, str);
  return (p);
}
```

A.A. 2001/2002

APA - Memoria dinamica

26

Struttura dati



A.A. 2001/2002

APA - Memoria dinamica

27

Deallocazione

Quando si desidera deallocare una zona di memoria, si usa la procedura di sistema `free`:

```
void free (void *);
```

La procedura `free` rende libera la zona di memoria puntata dal parametro, che deve essere stata precedentemente allocata con una chiamata a `malloc`.

A.A. 2001/2002

APA - Memoria dinamica

28



Esempio

Si vuole scrivere una procedura *libera*, che dealloca il vettore di *n* strutture passato come parametro.

È necessario deallocare anche la memoria usata per le stringhe.

A.A. 2001/2002

APA - Memoria dinamica

29



Procedura libera

```
void libera (struct scheda *p)
{
  int i;
  for (i=0; i<n; i++)
  {
    free (p[i].nome);
    free (p[i].cogn);
  }
  free (p);
}
```

A.A. 2001/2002


APA - Memoria dinamica

30



Procedura libera (vers. 2)

```
void libera (struct scheda *p)
{ int i; struct scheda *q;
  q=p;
  for (i=0; i<n; i++)
  { free (q->nome);
    free (q->cogn);
    q++;
  }
  free (p);
}
```



Soluzione più
efficiente